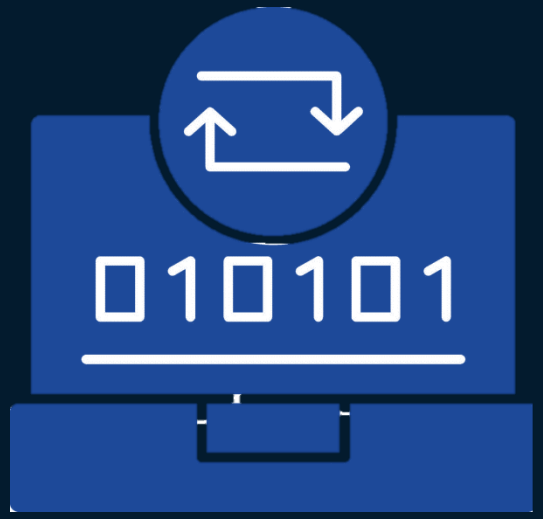
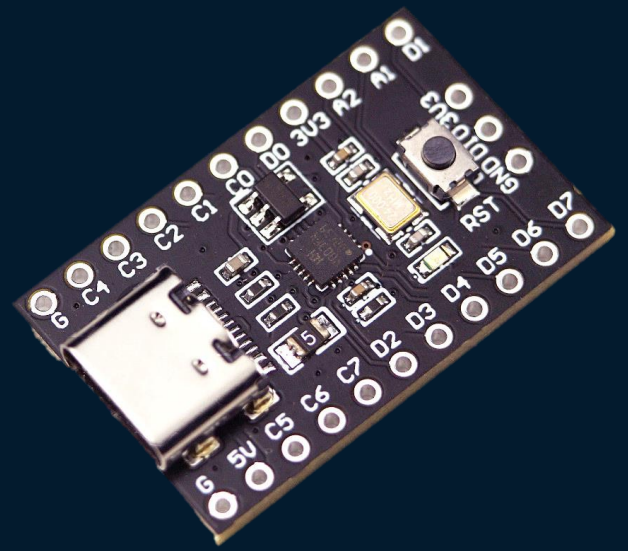
Embedded Systems

Day 5

Microprocessor Architecture
CH32V003



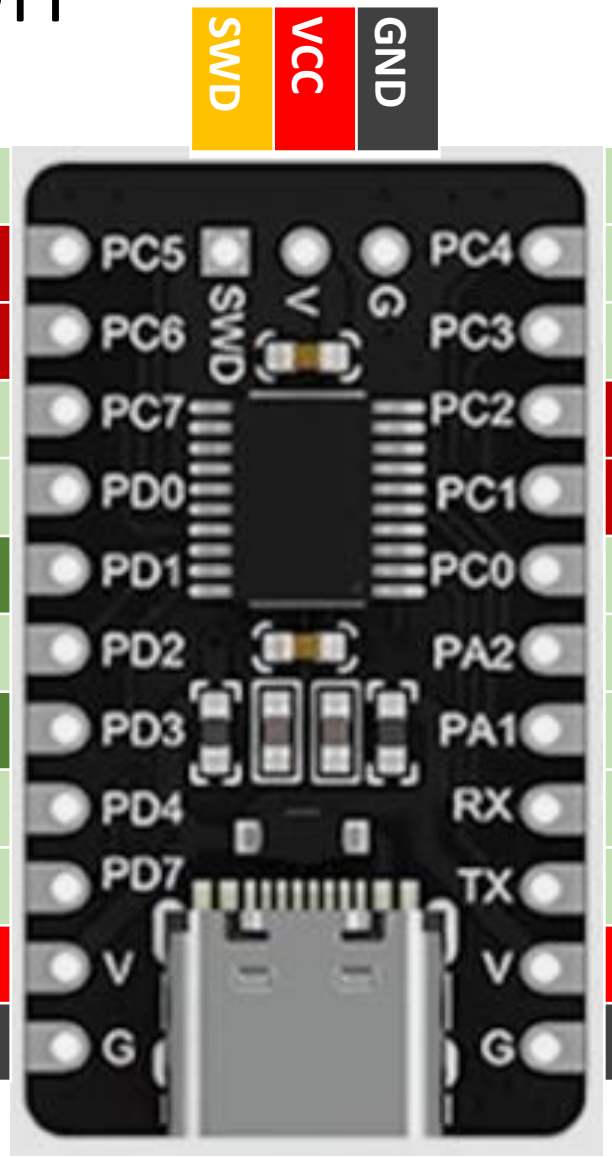


Contents

- Introduction to GPIOs (General Purpose Input/Output).
- Configuring GPIO pins as input and output.
- Basic GPIO operations: reading and writing to GPIO pins.
- Overview of communication protocols (UART, SPI, I2C).
- Analog to Digital Converter (ADC) and Digital to Analog Converter (DAC)
- Understanding the principles and advantages of each protocol.
- Basic implementation of communication protocols in embedded systems.

Pin Configuration

SCOM	TIM1	TIM2	ADC
SCK	T1ETR		5VT
MOSI			5VT
MISO			
	TICH1N	**SWIO**	
	T1CH3N		AETR2
	T1CH1		A3
UCTS		T2CH2	A4 AETR
UCK	**OPO**	T2CH1ETR	A7
OPP1	**NRST**	T2CH4	
			VCC
			GND

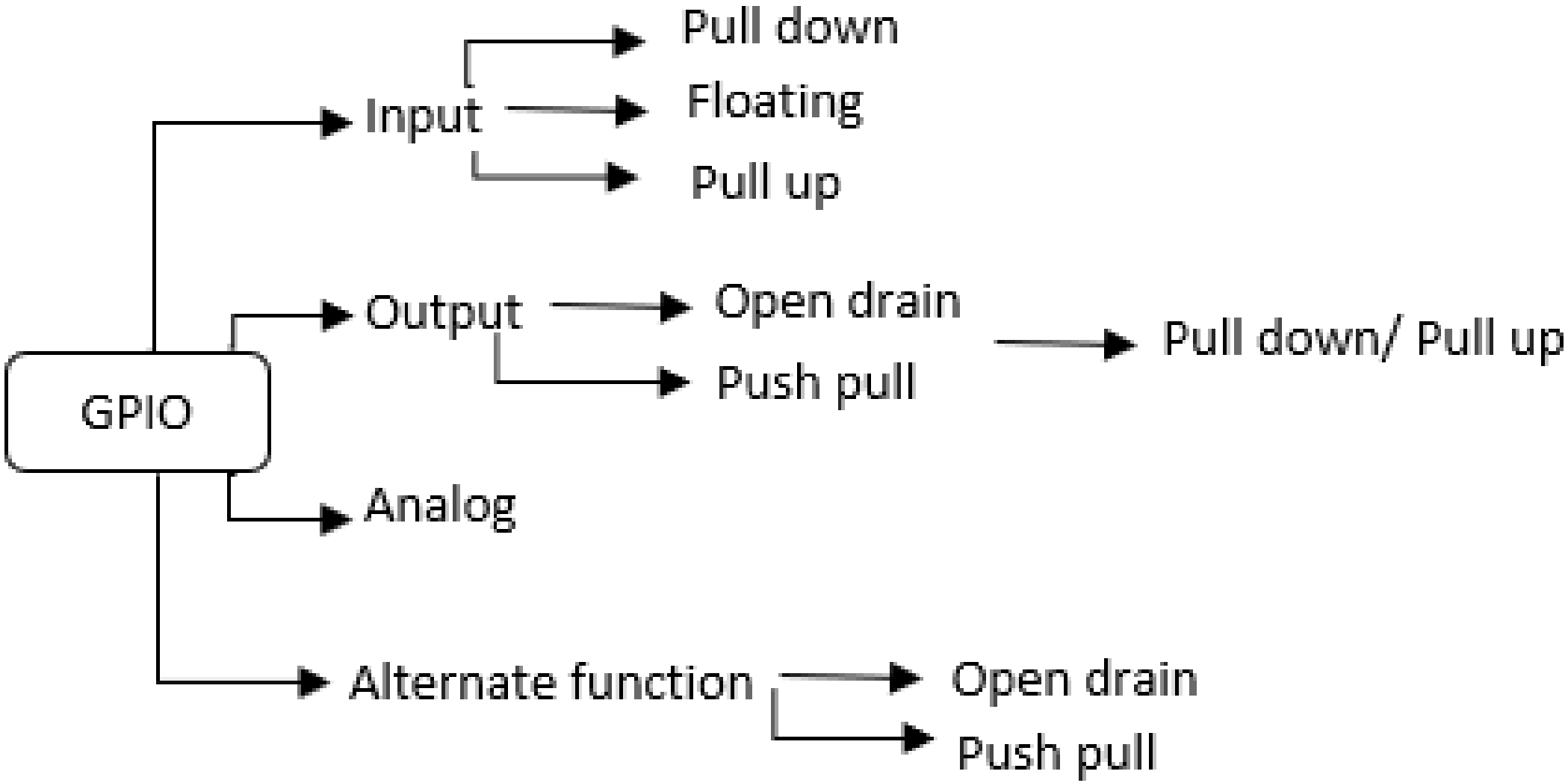


ADC	TIM1	TIM2	SCOM
A2	T1CH4	**MCO**	
	T1CH3		
5VT	T1BKIN		SCL URTS
5VT			SDA NSS
		T2CH3	
A0	TICH2N	**OPP0**	
A1	T1CH2	**OPN0**	
A6			URX
A5			UTX
VCC			
GND			

General Purpose Input Output Pins (GPIO)

- A GPIO (general-purpose input/output) port handles both incoming and outgoing digital signals.
- The GPIO port can be configured for multiple input or output modes, with built-in pull-up or pull-down resistors that can be turned off, and can be configured for push-pull or open-drain functions. the GPIO port can also be multiplexed for other functions.
- GPIO as Input: It can be used to communicate to the CPU the ON/OFF signals received from switches, or the digital readings received from sensors.
- As an Output port: It can be used to drive outside operations based on CPU instructions and calculation results—for example, to drive an LED display based on calculation results, or to output drive signals to a motor.
- In early MCUs, each port was either exclusively input or exclusively output.
- A GPIO is flexible, however. If it has 8 pins , you can set them as best suits your needs:
    - 4 input and 4 output
    - 7 input and 1 output
    - any other combination
- Note that while programs read, write, and operate on digital values (0s and 1s), external devices often use signal levels: LOW voltage and HIGH voltage. The GPIO handles the necessary conversions in both directions.

GPIO Functions

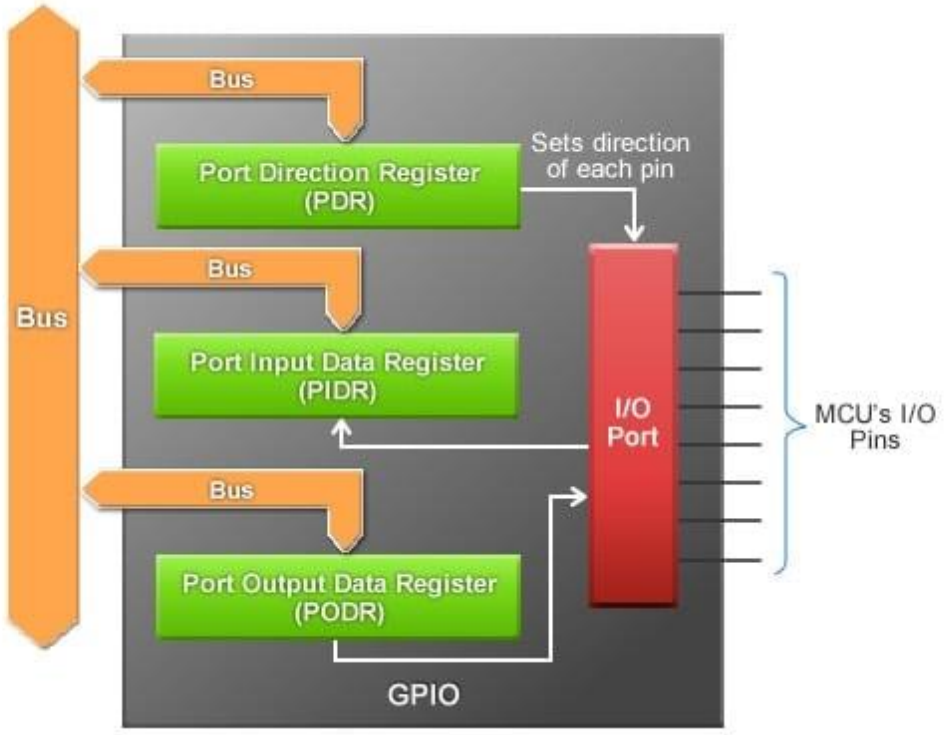


GPIO General Registers

- **Port Direction Register (PDR)**
  Sets the direction of each GPIO pin; either input or output.

- **Port Input Data Register (PIDR)**
  Shows status of the input pins. For each pin, input of a LOW signal sets the corresponding register value to 0; input of a HIGH signal sets the value to 1. The CPU reads this register in order to learn the most recent signal levels. Values are not saved; each time the CPU reads the register, it will reflect the current signal states.

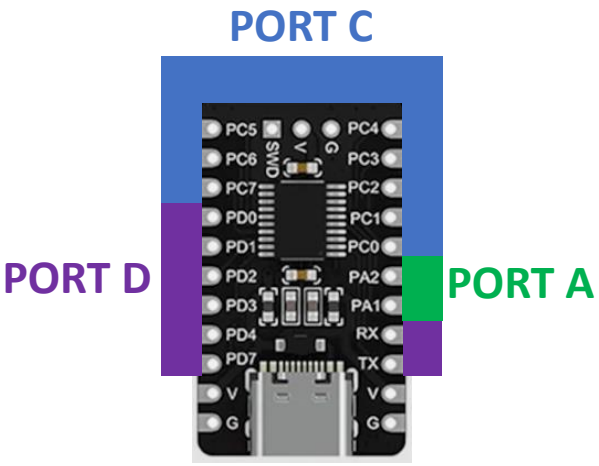
- **Port Output Data Register (PODR)**
  To output data through the output pins, the CPU writes the output values to the register. A value of 0 is converted into a LOW output; 1 is converted into HIGH output. As with regular memory, the values written here are retained until overwritten. This means that the pin output level will also be maintained until the value is changed.



GPIO Main Features in CH32V003

Each pin of the port can be configured to one of the following multiple modes.

- Floating input
- Pull-up input
- Dropdown input
- Analog input
- Open drain output
- Push-pull output
- Multiplexing the inputs and outputs of functions



Many pins have multiplexing capabilities, and many other peripherals map their output and input channels to these pins. The specific usage of these multiplexed pins needs to be referred to the individual peripherals, and the content of whether these pins are multiplexed and remapped is explained in this chapter.

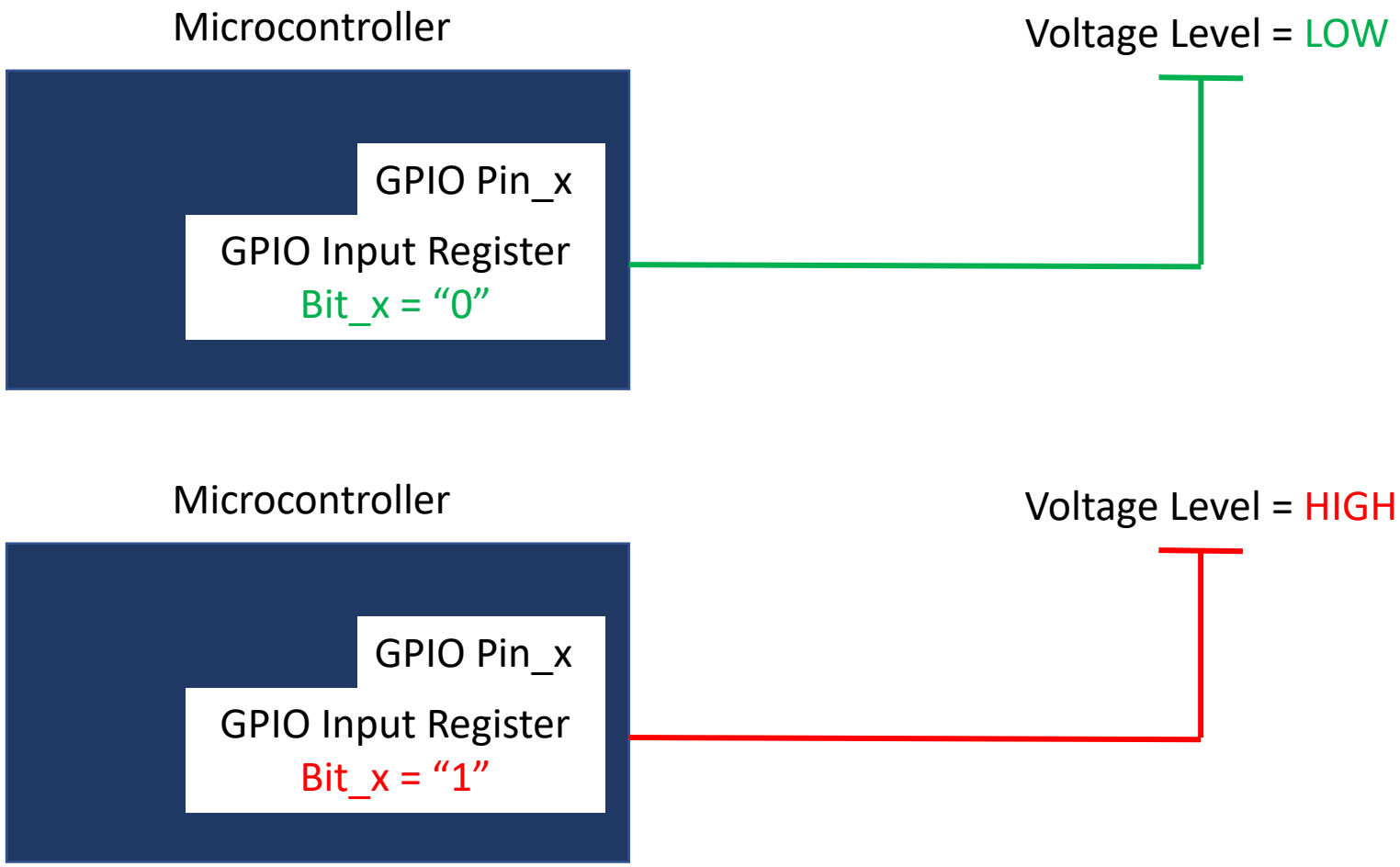
GPIO Ports:

There are three port groups available in the MCU, Port **A**, **C** and **D** and in each group, number of pins are there.

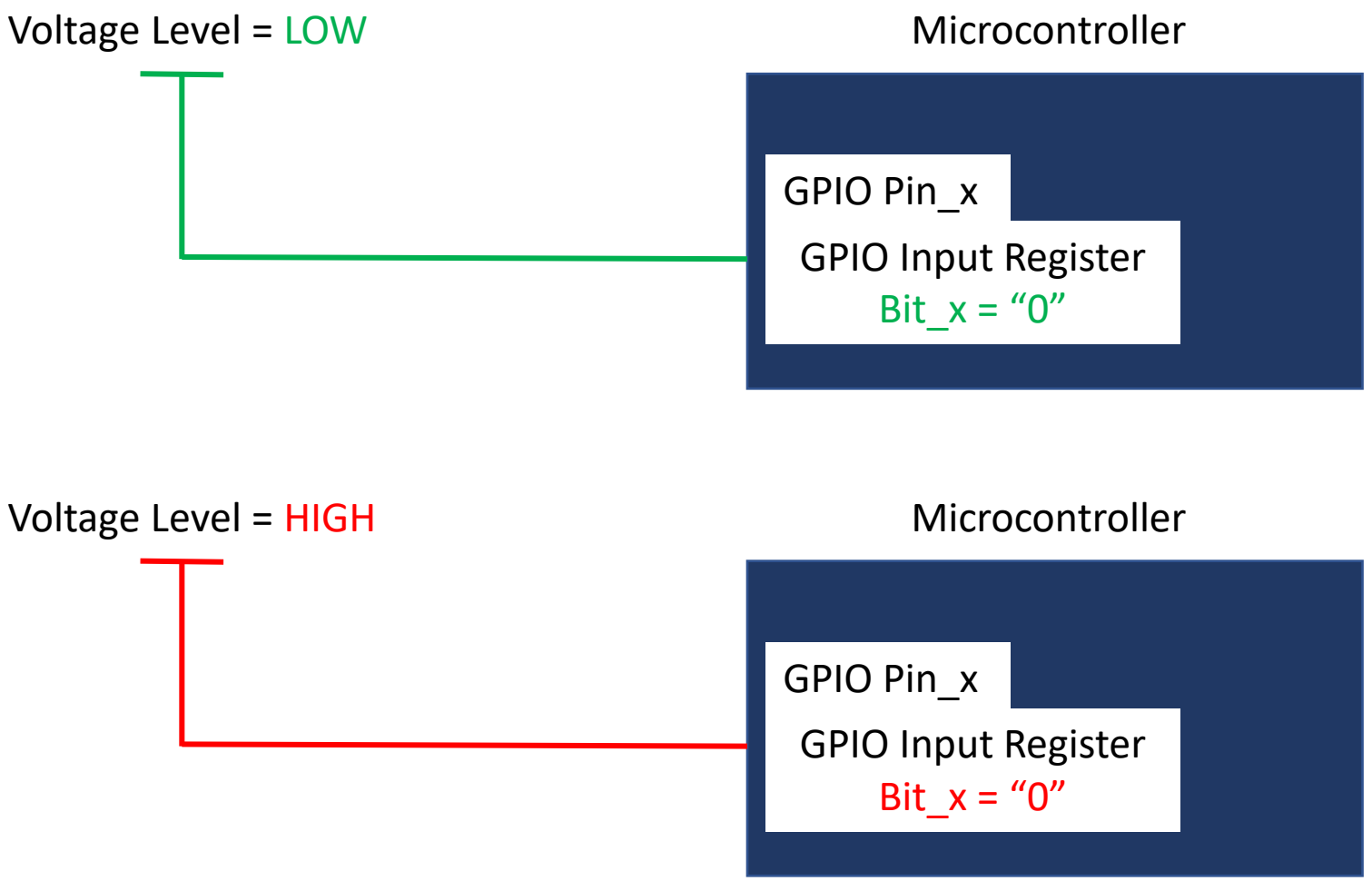
Each Port pin can be configured as GPIO output and can be used in the application.

II

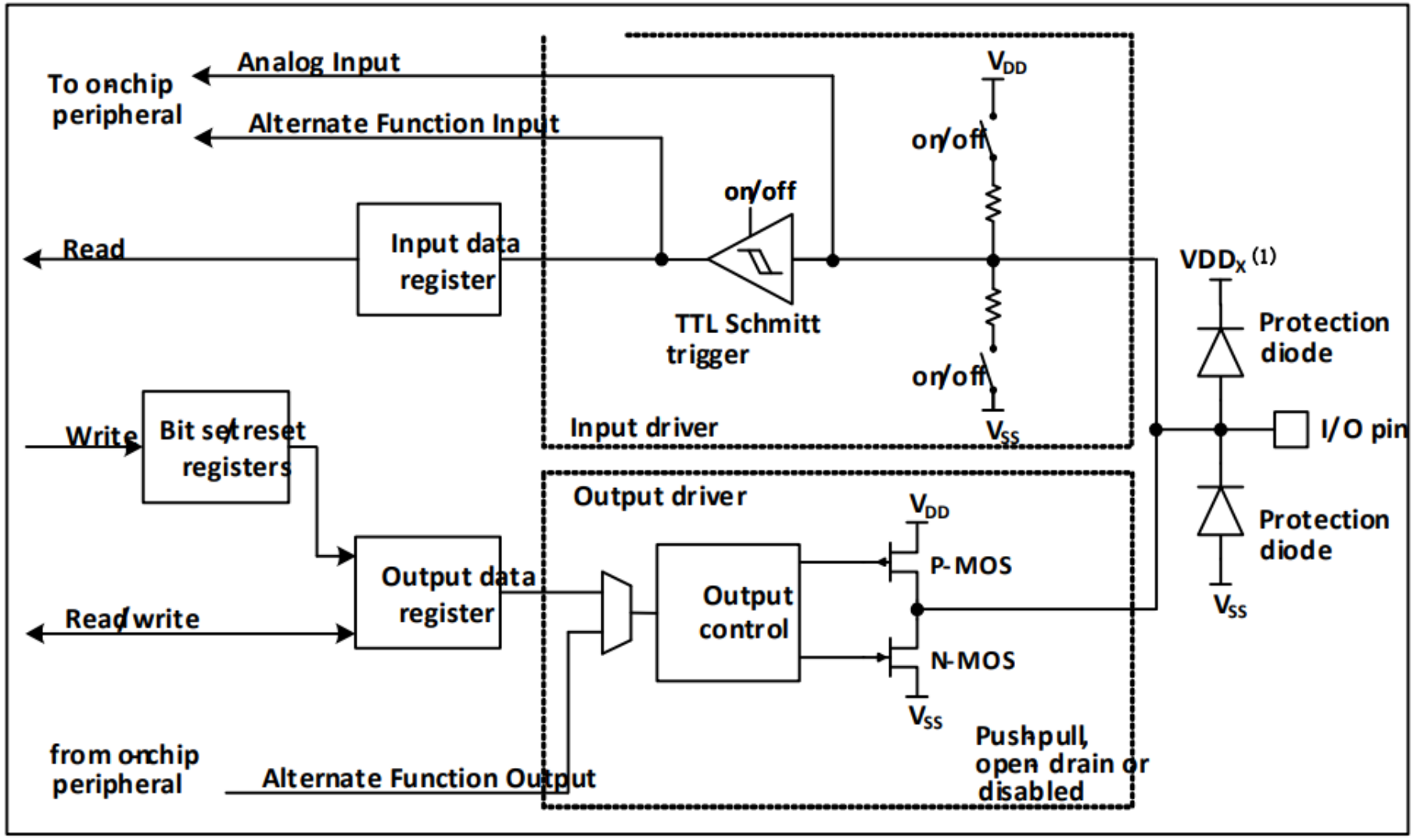
GPIO as Digital Output



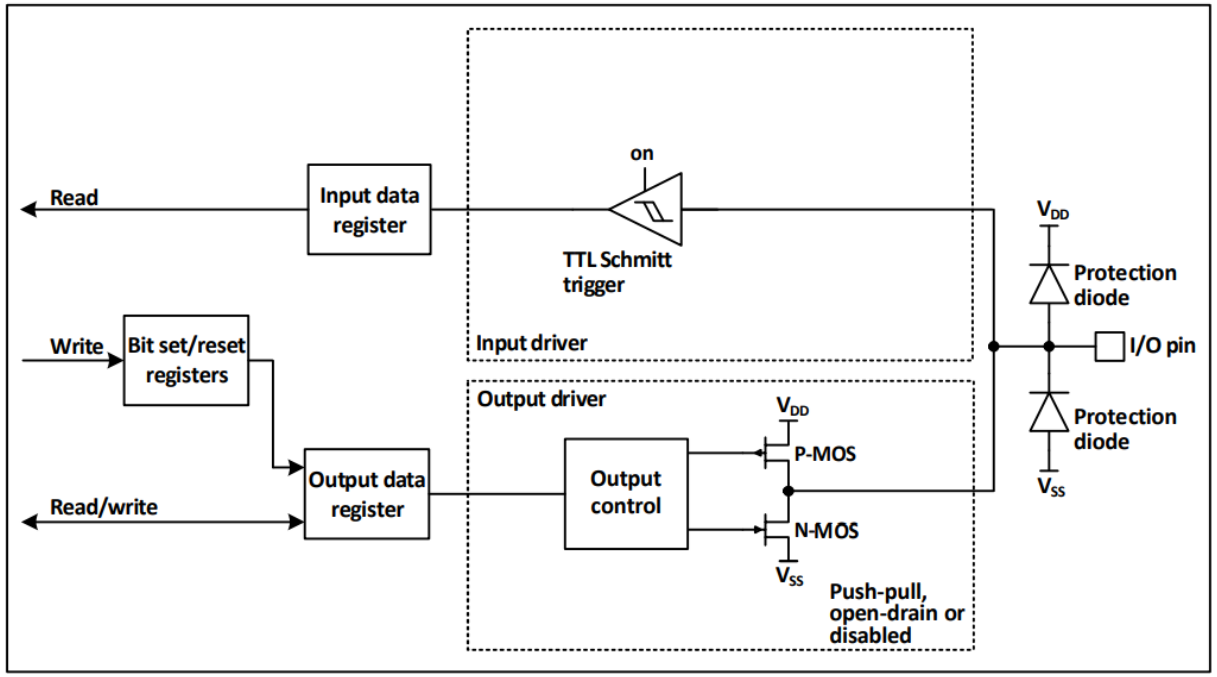
GPIO as Digital Input



GPIO Input/Output Circuit

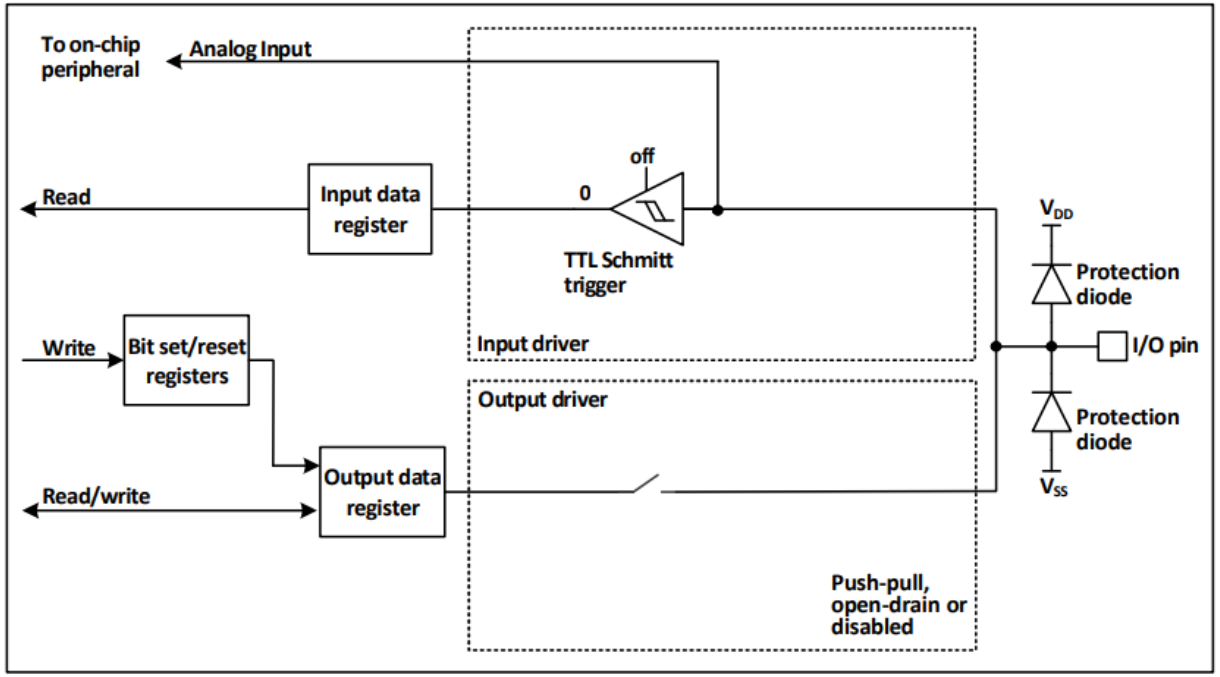


GPIO Output Circuit



When the IO port is configured to output mode, the pair of MOS in the output driver can be configured to push-pull or open-drain mode as needed, without using the multiplexing function. The pull-up and pull-down resistors of the input driver are disabled, the TTL Schmitt trigger is activated, and the levels appearing on the IO pins will be sampled into the input data registers at each AHB clock, so reading the input data registers will give the IO status, and in push-pull output mode, access to the output data registers will give the last written value.

Analog Input Configuration



When the analog input is enabled, the output buffer is disconnected, the input of the Schmitt trigger in the input driver is disabled to prevent the generation of consumption on the IO port, the pull-up and pull-down resistors are disabled, and the read input data register will always be 0.

GPIO Registers of CH32V003

- Unless otherwise specified, the registers of the GPIO must be operated as words (operate these registers with 32 bits).
- Each GPIO Port has its own Register-Set which includes these following registers

Register Name	Function
Port Configuration Register	Configures pin modes and settings.
Port Input Data Register	Reads the state of input pins.
Port Output Data Register	Sets the state of output pins.
Port Set/Reset Register	Allows setting or resetting individual output pins.
Port Reset Register	Used to reset configurations or outputs of the port.
Port Configuration Lock Register	Controls the locking and unlocking of port configuration settings.

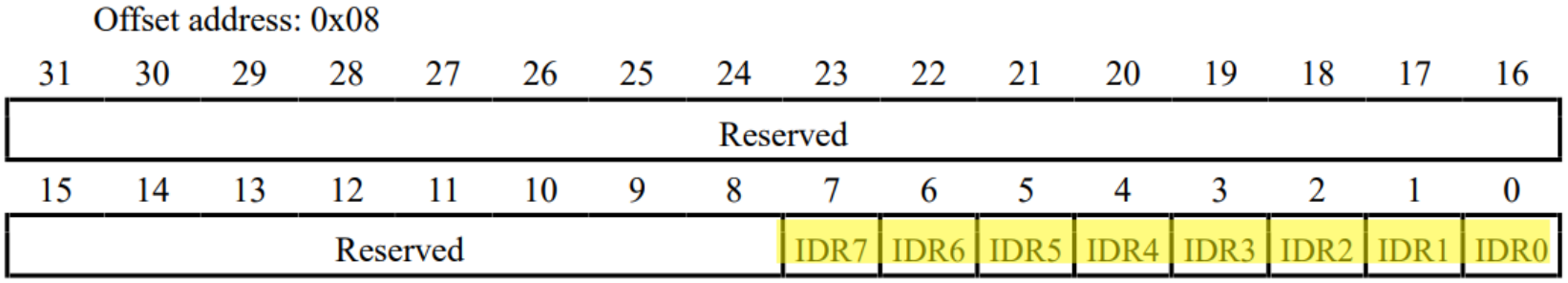
Port Configuration Register Low (GPIOx_CFGLR) (x=A/C/D)

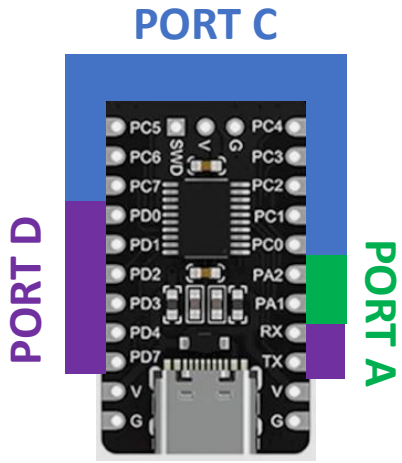
Offset address: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	

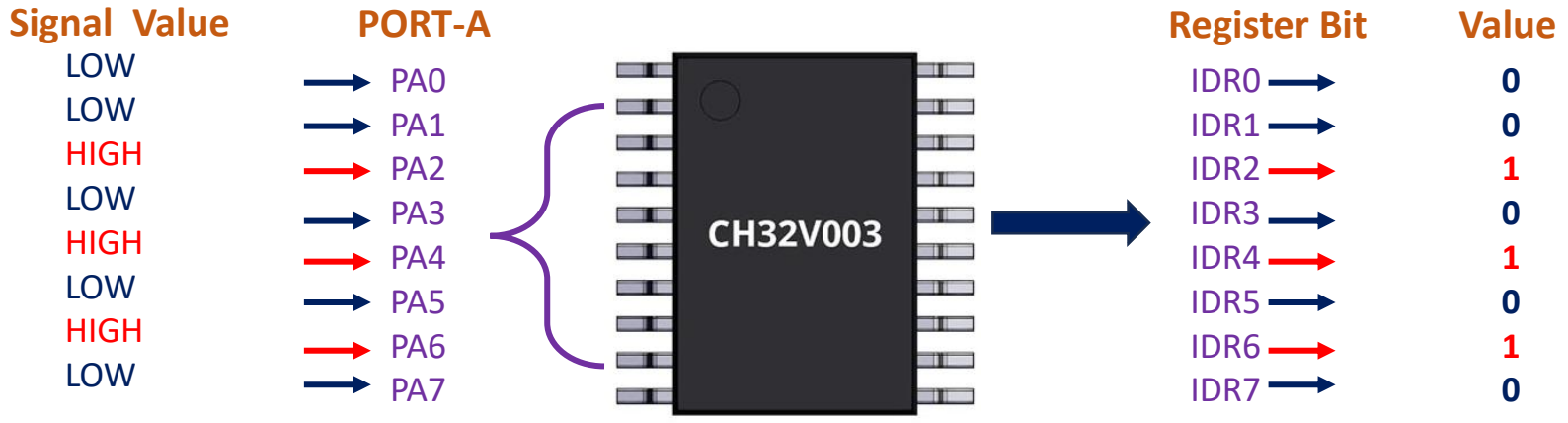
Bits	Name	Access	Description	RV
[31:30] [27:26] [23:22] [19:18] [15:14] [11:10] [7:6] [3:2]	CNFy[1:0]	RW	(y=0-7), the configuration bits for port x, by which the corresponding port is configured. When in input mode (MODE=00b). ▪ 00: Analog input mode. ▪ 01: Floating input mode. ▪ 10: With pull-up and pull-down mode. ▪ 11: Reserved. In output mode (MODE>00b). ▪ 00: Universal push-pull output mode. ▪ 01: Universal open-drain output mode. ▪ 10: Multiplexed function push-pull output mode. ▪ 11: Multiplexing function open-drain output mode.	01b
[29:28] [25:24] [21:20] [17:16] [13:12] [9:8] [5:4] [1:0]	MODEy[1:0]	RW	(y=0-7), port x mode selection, configure the corresponding port by these bits. • 00: Input mode. • 01: Output mode, maximum speed 10MHz; • 10: Output mode, maximum speed 2MHz. • 11: Output mode, maximum speed 50MHz	00b

Port Input Register (GPIOx_INDR) (x=A/C/D)

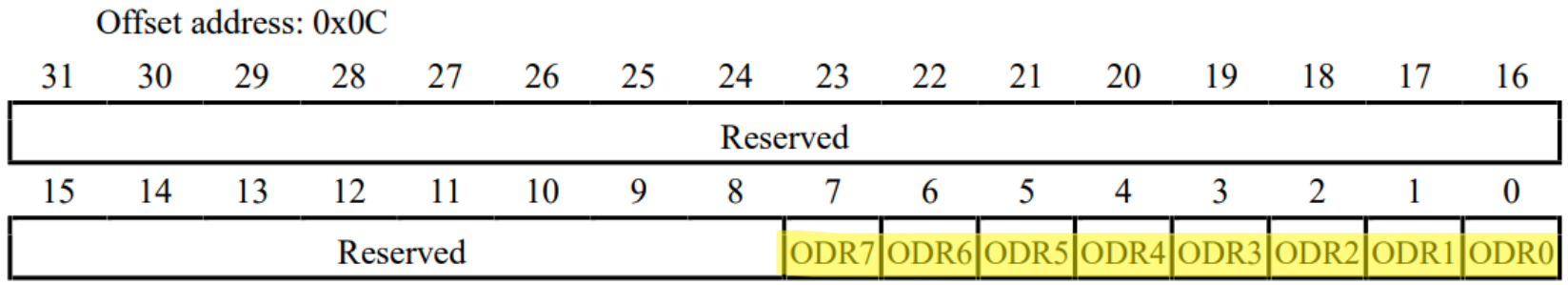


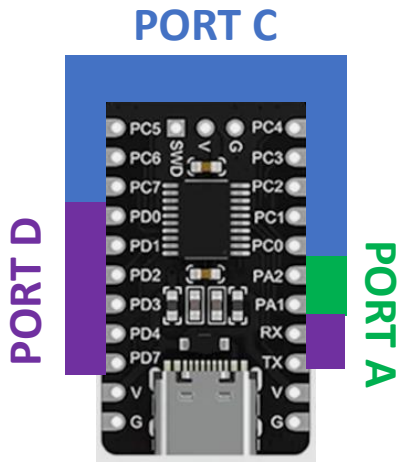


Bits	Name	Access	Description	RV
[31:8]	Reserved	RO	-	0b
[7:0]	IDRy	RO	(y=0-7), the port input data. These bits are **readonly** and can only be read out in 16-bit form. The value read is the high and low state of the corresponding bit.	0b



Port Output Register (GPIOx_ODR) (x=A/C/D)



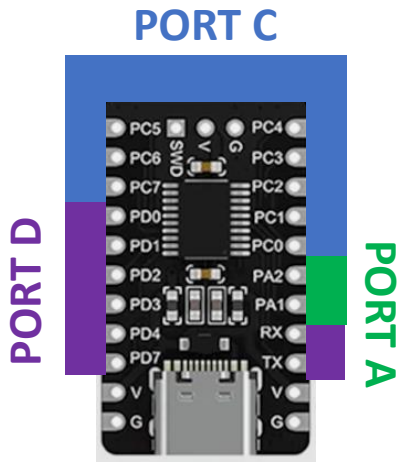


Bits	Name	Access	Description	RV
[31:8]	Reserved	RO	-	0b
[7:0]	ODRy	RW	For output modes. (y=0-7), the data output by the port. These data can only be operated in 16-bit form. the I/O port outputs the values of these registers externally. For modes with drop-down inputs. 0: Drop-down input. 1: Pull-up input.	0b

Port Reset/Set Register (GPIOx_BCR) (x=A/C/D)

Offset address: 0x10

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0



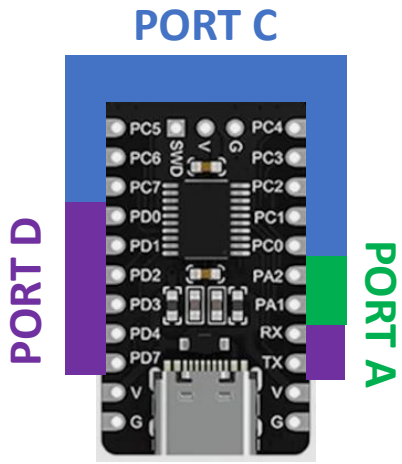
Bits	Name	Access	Description	RV
[31:24]	Reserved	RO	-	
[23:16]	BRy	RW	(y=0-7), the corresponding OUTDR bits are cleared for these location bits, and writing 0 has no effect. These bits can only be accessed in 16-bit form. If both BR and BS bits are set, the BS bit takes effect	0b
[15:8]	Reserved	RO	-	
[7:0]	BSy	RW	(y=0-7), for which the location bits will make the corresponding OUTDR location bits, writing 0 has no effect. These bits can only be accessed in 16-bit form. If both BR and BS bits are set, the BS bit takes effect.	0b

Port Reset Register (GPIOx_BCR) (x=A/C/D)

Offset address: 0x14

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0



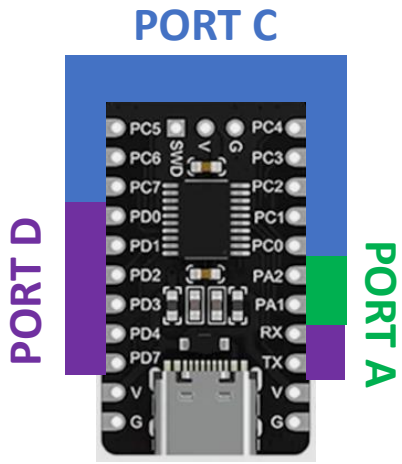
Bits	Name	Access	Description	RV
[31:8]	Reserved	RO	-	
[7:0]	BRy	RW	(y=0-7), the corresponding OUTDR bits are cleared for these location bits, and writing 0 has no effect. These bits can only be accessed in 16-bit form.	0b

Port Reset Register (GPIOx_BCR) (x=A/C/D)

Offset address: 0x14

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

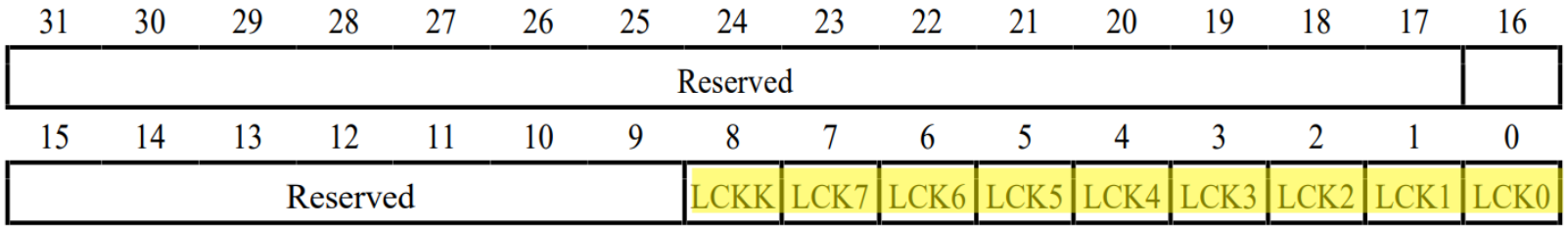
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0



Bits	Name	Access	Description	RV
[31:8]	Reserved	RO	-	
[7:0]	BRy	RW	(y=0-7), the corresponding OUTDR bits are cleared for these location bits, and writing 0 has no effect. These bits can only be accessed in 16-bit form.	0b

Port Configuration Lock Register (GPIOx_LCKR) (x=A/C/D)

Offset address: 0x18



Bits	Name	Access	Description	RV
[31:8]	Reserved	RO	-	
8		RW	The lock key, which can be written in a specific sequence to achieve locking, but which can be read out at any time. It reads 0 to indicate that no locking is in effect, and reads 1 to indicate that locking is in effect. The write sequence for the lock key is: write 1 - write 0 - write 1 - read 0 - read 1. The last step is not necessary, but can be used to confirm that the lock key is active. Any error while writing the sequence will not enable the activation of the lock and the value of LCK[7:0] cannot be changed while the sequence is being written. After the lock is in effect, the port configuration can only be changed after the next reset.sponding OUTDR bits are cleared for these location bits, and writing 0 has no effect. These bits can only be accessed in 16-bit form.	0b
[7:0]	LCKy		(y=0-7), these bits are 1 to indicate locking the configuration of the corresponding port. These bits can only be changed before the LCKK is unlocked. The locked configuration refers to the configuration registers GPIOx_CFGLR and GPIOx_CFGHR.	0b

Note: After the LOCK sequence is executed for the corresponding port bit, the configuration of the port bit will not be changed again until the next system reset.

Alternate Functions

- In microcontrollers, alternate functions refer to the capability of pins to serve multiple purposes depending on the configuration.
- These functions enable a single physical pin to be used for different roles such as digital input/output, analog input, communication interfaces (like UART, SPI, I2C), timers, and more.

Purpose of Alternate Functions

1	**Pin Multiplexing:**	Microcontrollers often have a limited number of pins. Alternate functions allow these pins to be multiplexed to serve various roles, thus maximizing the functionality of the microcontroller.
2	**Flexibility in Design:**	Designers can choose which peripheral functions are mapped to which pins, providing flexibility in PCB layout and design.
3	**Resource Optimization:**	By allowing pins to be used for different purposes, alternate functions help in optimizing the use of available hardware resources.

Alternate Functions

- In microcontrollers, alternate functions refer to the capability of pins to serve multiple purposes depending on the configuration.
- These functions enable a single physical pin to be used for different roles such as digital input/output, analog input, communication interfaces (like UART, SPI, I2C), timers, and more.

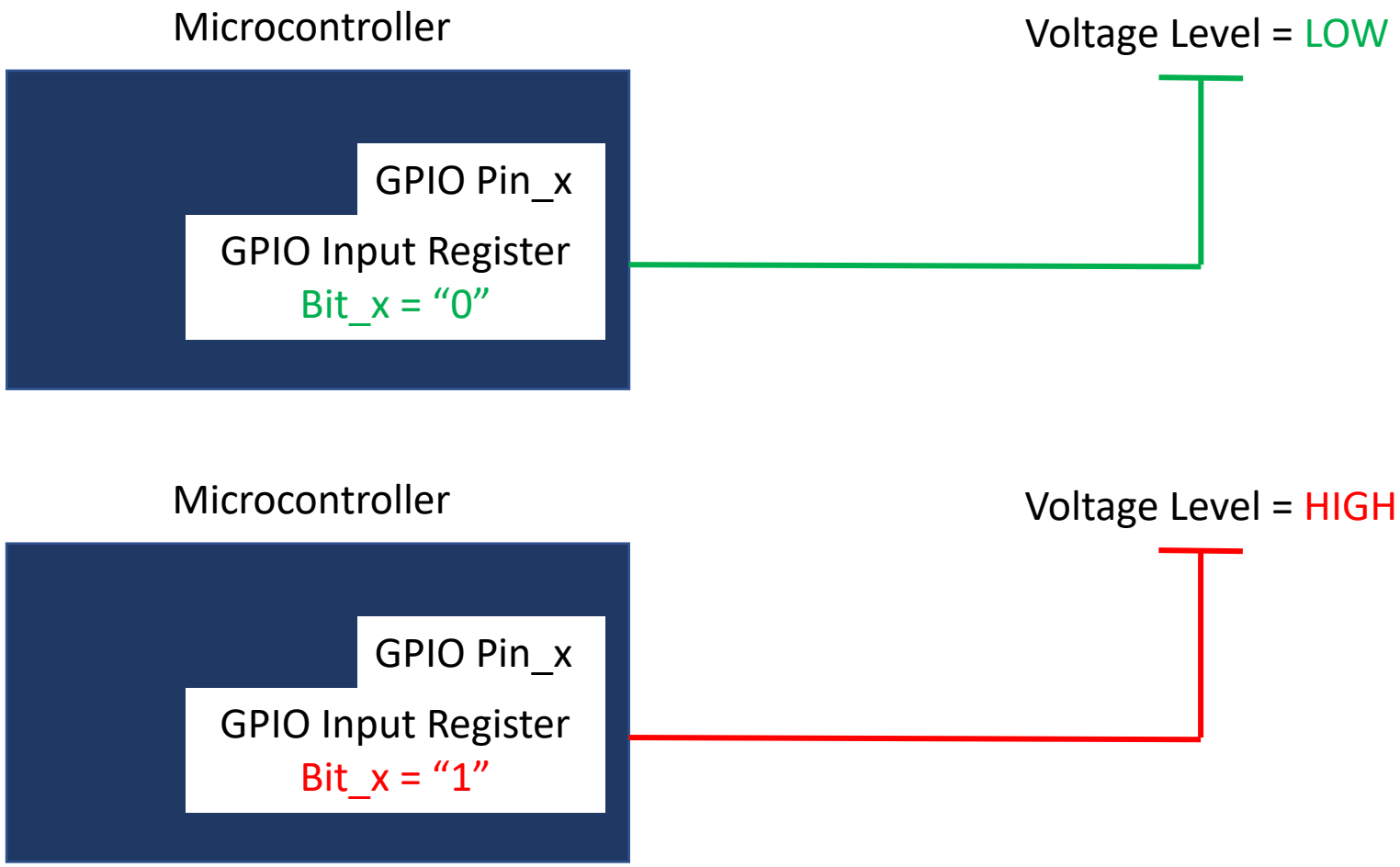
Purpose of Alternate Functions

1	**Pin Multiplexing:**	Microcontrollers often have a limited number of pins. Alternate functions allow these pins to be multiplexed to serve various roles, thus maximizing the functionality of the microcontroller.
2	**Flexibility in Design:**	Designers can choose which peripheral functions are mapped to which pins, providing flexibility in PCB layout and design.
3	**Resource Optimization:**	By allowing pins to be used for different purposes, alternate functions help in optimizing the use of available hardware resources.

Common Alternate Functions

GPIO | Timers and Counters | Analog Functions | Communication

GPIO as Digital Output



Setting GPIOs as Output: Steps

GPIO Initialization Steps

1. For any configuration, **clock** for that **GPIO port** needs to be enabled first.
2. After reset GPIOs are they run in their initial states and most of them run in **floating states**.
3. Initialize the GPIOs according to the **desired functionality**.
4. Define these **parameters** for initializing the GPIO pin that will be described in the below example.

Three Parameters are there for any GPIO when configuring as output

	Parameters	Descrption
1	**GPIO_Pin:**	this is to define which Pin, for example for D0 it will be GPIO_Pin_0
2	**GPIO_Mode:**	to configure if the output will be an open drain (GPIO_Mode_Out_OD) or push-pull (GPIO_Mode_Out_PP).
3	**GPIO_Speed:**	to configure how fast you want control the GPIO. There are three options: GPIO_Speed_**10MHz**, GPIO_Speed_**2MHz**, GPIO_Speed_**50MHz**. This basically configures the drive strength of the GPIO internally.

Setting GPIOs as Output: Steps

It is recommended to read the datasheet *and go through the electrical characteristics section to know more about capabilityand limitation of GPIO port/pins. Like output current sourcing and sinking capability, etc.*

Please note PD7 is by default is configured as MCU reset pin, you need to configure it as GPIO by configuring it with WCH Link Programmer Utility or in the code.

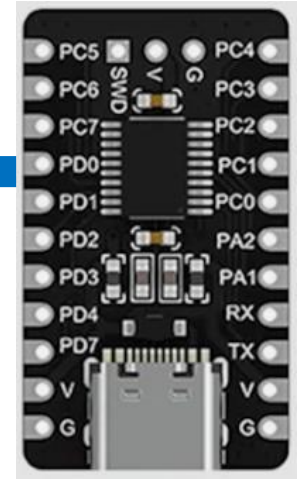
Setting GPIOs as Output: Example Code

GPIO as Output Example Code for CH32V003:

Single GPIO Pin

```
1. void GPIO_Config (void)
2. {
3.  GPIO_InitTypeDef GPIO_InitStructure = {0}; //structure variable used for the GPIO
   Configuration
4. RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE); // to Enable the clock for Port D
5. GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; // Defines which Pin to configure
6. GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Defines Output Type
7. GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; // Defines speed
8. GPIO_Init(GPIOD, &GPIO_InitStructure);
9.  }
```

GPIO PIN: **PD0**
GPIO MODE: **Output**
GPIO Speed: **50 MHz**



Setting GPIOs as Output: Example Code

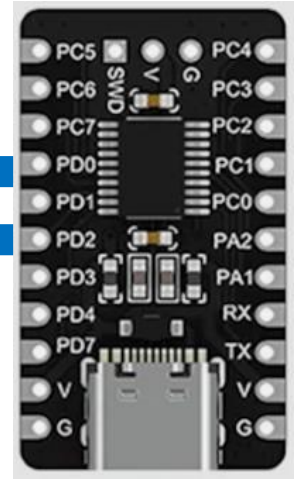
GPIO as Output Example Code for CH32V003:

Single GPIO Pin

If **multiple pins** of same port need to be configured with similar settings, you can write as shown below (GPIO D0 and D1):

```
1. void GPIO_Config(void)
2. {
3.  GPIO_InitTypeDef GPIO_InitStructure = {0}; //structure variable used for the GPIO
 configuration
4. RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE); // to Enable the clock for Port D
5. GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_PIn_2; // Defines which Pin to configure
6. GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Defines Output Type
7. GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; // Defines speed
8. GPIO_Init(GPIOD, &GPIO_InitStructure); }
```

GPIO PIN: **PD0,PD2**
GPIO MODE: **Output**
GPIO Speed: **50 MHz**



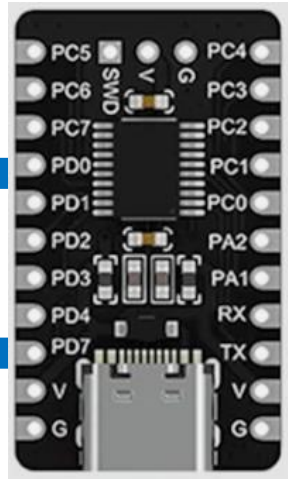
Setting GPIOs as Output: Example Code

But, if you have **different settings** for different GPIOs of same port you can do like this (GPIO D0 and D7):

```
1. void GPIO_Config(void)
2. {
3.  GPIO_InitTypeDef GPIO_InitStructure = {0}; //structure variable used for the GPIO
 configuration
4. RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE); // to Enable the clock for Port D
5. GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; // Defines which Pin to configure
6. GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Defines Output Type
7. GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; // Defines speed
8.
9. GPIO_Init(GPIOD, &GPIO_InitStructure);
10.GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1; // Defines which Pin to configure
11.GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD; // Defines Output Type
12.GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; // Defines speed
13.
14.GPIO_Init(GPIOD, &GPIO_InitStructure);
15.}
```

GPIO PIN: **PD0**
GPIO MODE: **Output**
GPIO Speed: **50 MHz**

GPIO PIN: **PD7**
GPIO MODE: **Output**
GPIO Speed: **2 MHz**



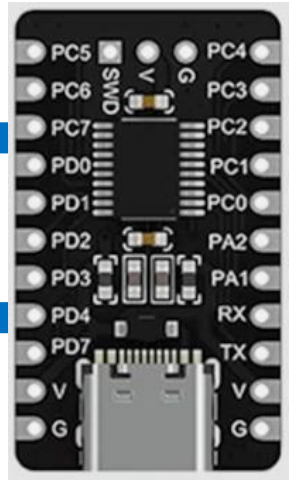
Setting GPIOs as Output: Example Code

And, now suppose if you want to configure different configuration of same ports, you can write code like this: (GPIO D4 and C7).

```
1. void GPIO_Config(void)
2. {
3.  GPIO_InitTypeDef GPIO_InitStructure = {0}; //structure variable used for the GPIO
 configuration
4. RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE); // to Enable the clock for Port D
5. GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4; // Defines which Pin to configure
6. GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Defines Output Type
7. GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; // Defines speed
8.
9. GPIO_Init(GPIODC &GPIO_InitStructure);
10.GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7; // Defines which Pin to configure
11.GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD; // Defines Output Type
12.GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; // Defines speed
13.
14.GPIO_Init(GPIOC, &GPIO_InitStructure);
15.}
```

GPIO PIN: **PC7**
GPIO MODE: **Output**
GPIO Speed: **2 MHz**

GPIO PIN: **PD4**
GPIO MODE: **Output**
GPIO Speed: **50 MHz**



Setting GPIOs as Output: Functions

let us see which all functions are available for controlling the GPIO pins or port.

If you go through ch32v00x_gpio.h header file, you can see there following function which you will be using for GPIO output operations

Functions

```
1. void     GPIO_SetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);

2. void     GPIO_ResetBits(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
3. void     GPIO_WriteBit(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
4. void     GPIO_Write(GPIO_TypeDef *GPIOx, uint16_t PortVal);
5. void     GPIO_PinLockConfig(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
```

Setting GPIOs as Output: GPIO Toggle

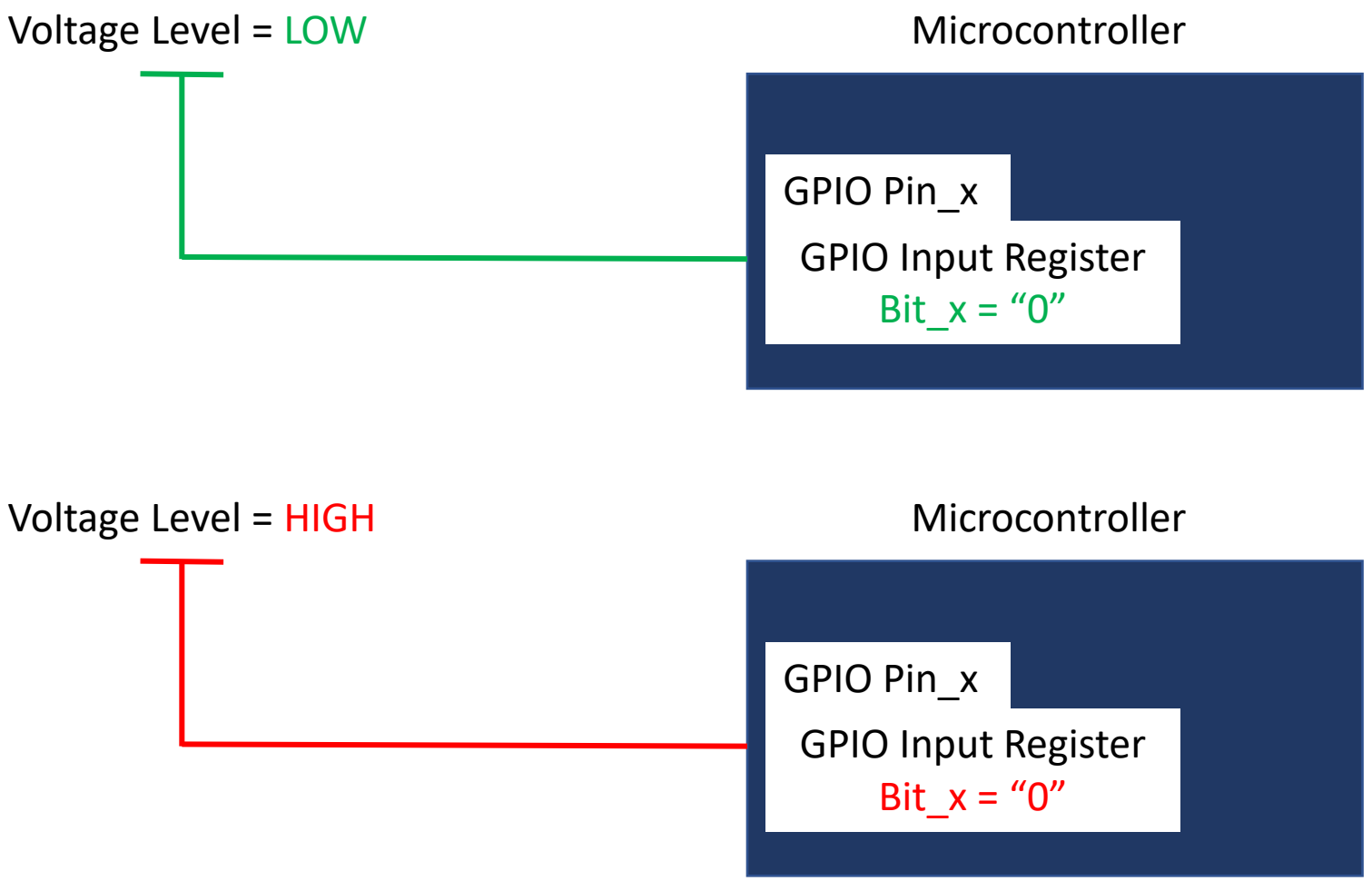
```
1. #include "debug.h"
2.
3. /*********************************************************************
4.  * @fn      GPIO_Toggle_INIT
5.  * @brief   Initializes GPIOA.0
6.  * @return  none
7.  */
8. void GPIO_Toggle_INIT(void)
9. {
10.     GPIO_InitTypeDef GPIO_InitStructure = {0};
11.
12.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
13.     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
14.     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
15.     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
16.     GPIO_Init(GPIOD, &GPIO_InitStructure);
17.}
19./*********************************************************************
20. * @fn      main
21. * @brief   Main program.
22. * @return  none
23. */
```

Setting GPIOs as Output: GPIO Toggle

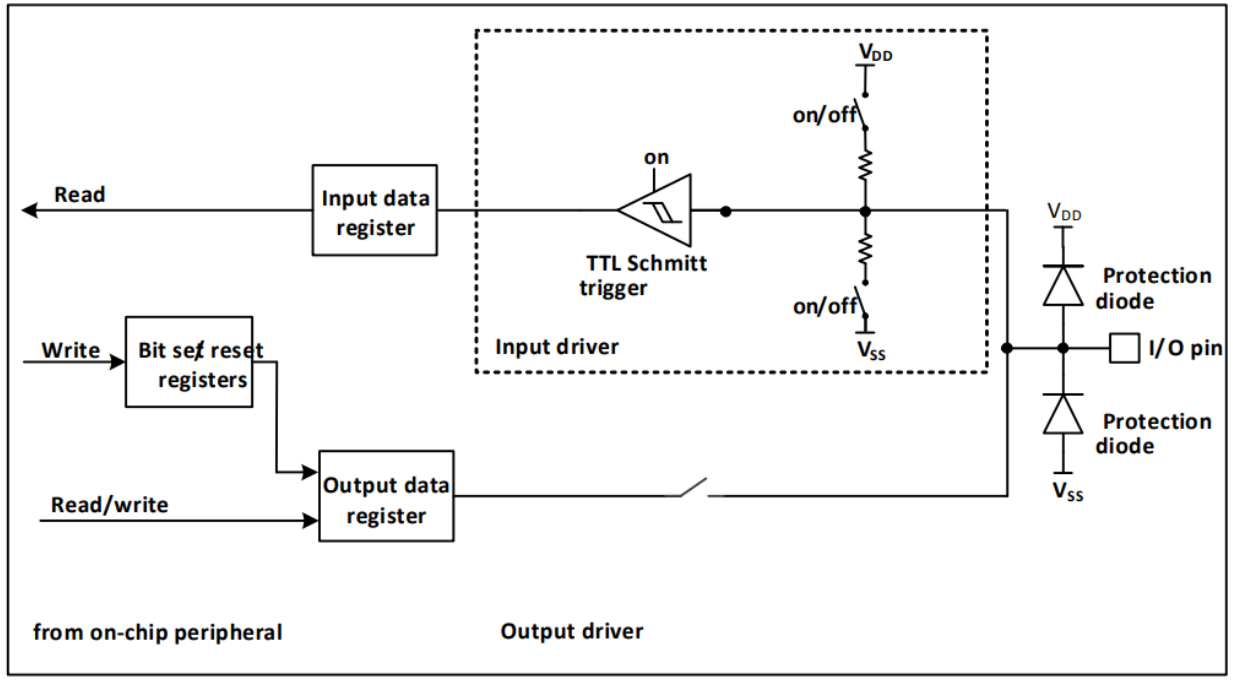
```
19. int main(void)
20. {
21.     u8 i = 0;
22.
23.     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
24.     SystemCoreClockUpdate();
25.     Delay_Init();
26. #if (SDI_PRINT == SDI_PR_OPEN)
27.     SDI_Printf_Enable();
28. #else
29.     USART_Printf_Init(115200);
30. #endif
31.     printf("SystemClk:%d\r\n", SystemCoreClock);
32.     printf( "ChipID:%08x\r\n", DBGMCU_GetCHIPID() );
33.     printf("GPIO Toggle TEST\r\n");
34.
35.     GPIO_Toggle_INIT();
36.
37.     while(1)
38.     {
39.         Delay_Ms(250); //SET THE DELAY VALUE HERE
40.         GPIO_WriteBit(GPIOD, GPIO_Pin_0, (i == 0) ? (i = Bit_SET) : (i = Bit_RESET));
41.     }
42. }
```

Setting GPIOs as Output: GPIO Toggle

GPIO as Digital Input



GPIO Input Circuit



When the IO port is configured in input mode, the output driver is disconnected, the input pull-up and pulldown are selectable, and no multiplexed functions or analog inputs are connected. The data on each IO port is sampled into the input data register at each AHB clock, and the level status of the corresponding pin is obtained by reading the corresponding bit of the input data register.

GPIO Input Polling vs Interrupt

GPIO Initialization Steps

1. For any configuration, **clock** for that **GPIO port** needs to be enabled first.
2. After reset GPIOs are they run in their initial states and most of them run in **floating states**.
3. Initialize the GPIOs according to the **desired functionality**.
4. Define these **parameters** for initializing the GPIO pin that will be described in the below example.

Three Parameters are there for any GPIO when configuring as output

	Parameters	Descrption
1	**GPIO_Pin:**	This is to define which Pin, for example for D0 it will be GPIO_Pin_0
2	**GPIO_Mode:**	To configure if the Input will be with internal pull-up resistance(GPIO_Mode_IPU)
3	**GPIO_Speed:**	to configure how fast you want control the GPIO. There are three options: GPIO_Speed_**10MHz**, GPIO_Speed_**2MHz**, GPIO_Speed_**50MHz**. This basically configures the drive strength of the GPIO internally.

Setting GPIOs as Digital Input: Steps

GPIO Initialization Steps

1. For any configuration, **clock** for that **GPIO port** needs to be enabled first.
2. After reset GPIOs are they run in their initial states and most of them run in **floating states**.
3. Initialize the GPIOs according to the **desired functionality**.
4. Define these **parameters** for initializing the GPIO pin that will be described in the below example.

Three Parameters are there for any GPIO when configuring as output

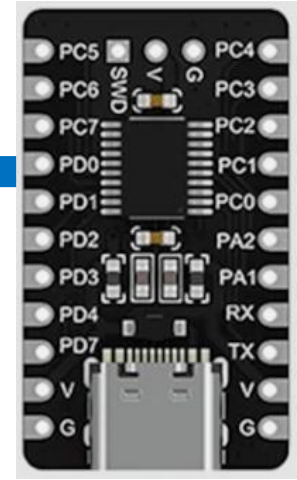
	Parameters	Descrption
1	**GPIO_Pin:**	This is to define which Pin, for example for D0 it will be GPIO_Pin_0
2	**GPIO_Mode:**	To configure if the Input will be with internal pull-up resistance(GPIO_Mode_IPU)
3	**GPIO_Speed:**	to configure how fast you want control the GPIO. There are three options: GPIO_Speed_**10MHz**, GPIO_Speed_**2MHz**, GPIO_Speed_**50MHz**. This basically configures the drive strength of the GPIO internally.

Setting GPIOs as Input Polling: Example Code

GPIO as Input Polling: Example Code for CH32V003:

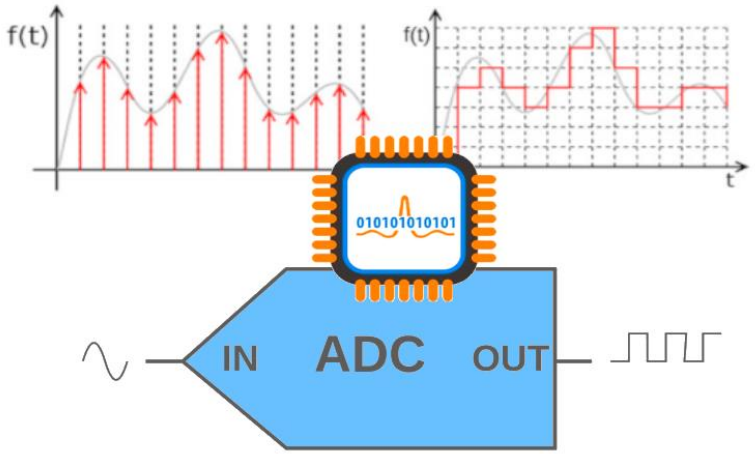
Single GPIO Pin

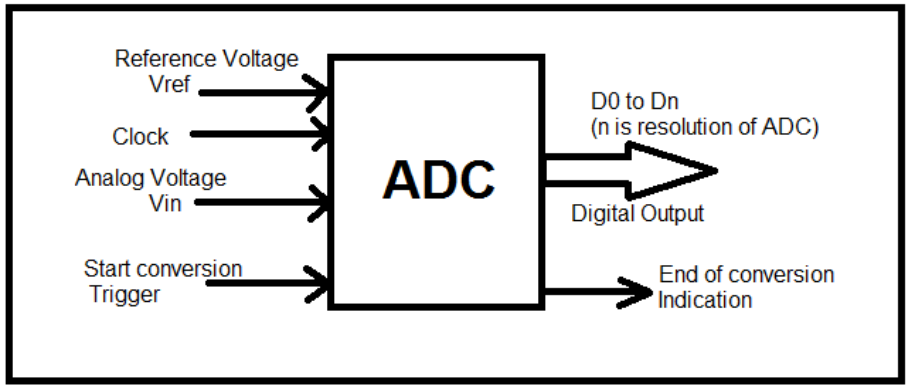
GPIO PIN: **PD0**
GPIO MODE: **Input**
GPIO Speed: **50 MHz**



Analog to Digital Converter

- For an ADC **input is analog signal** and **output is digital value correspond to analog signal**.
- To convert analog signal to digital ADC took help of **reference voltage i.e. some known voltage against which ADCs internal circuitry can compare input signal to calculate output digital value**.
- **Clock is required for sampling of input data.**
- Star of conversion is either hardware based or software based to star
- ADC and there will be end of conversion signal flag or interrupt.





Resolution	Min Voltage increment
8-bit	3.92 mV
10-bit	0.98 mV
12-bit	0.244 mV
14-bit	61 µV
16-bit	15 µV

Some Critical Parameters in Analog to Digital Conversion

Sampling Rate of ADC

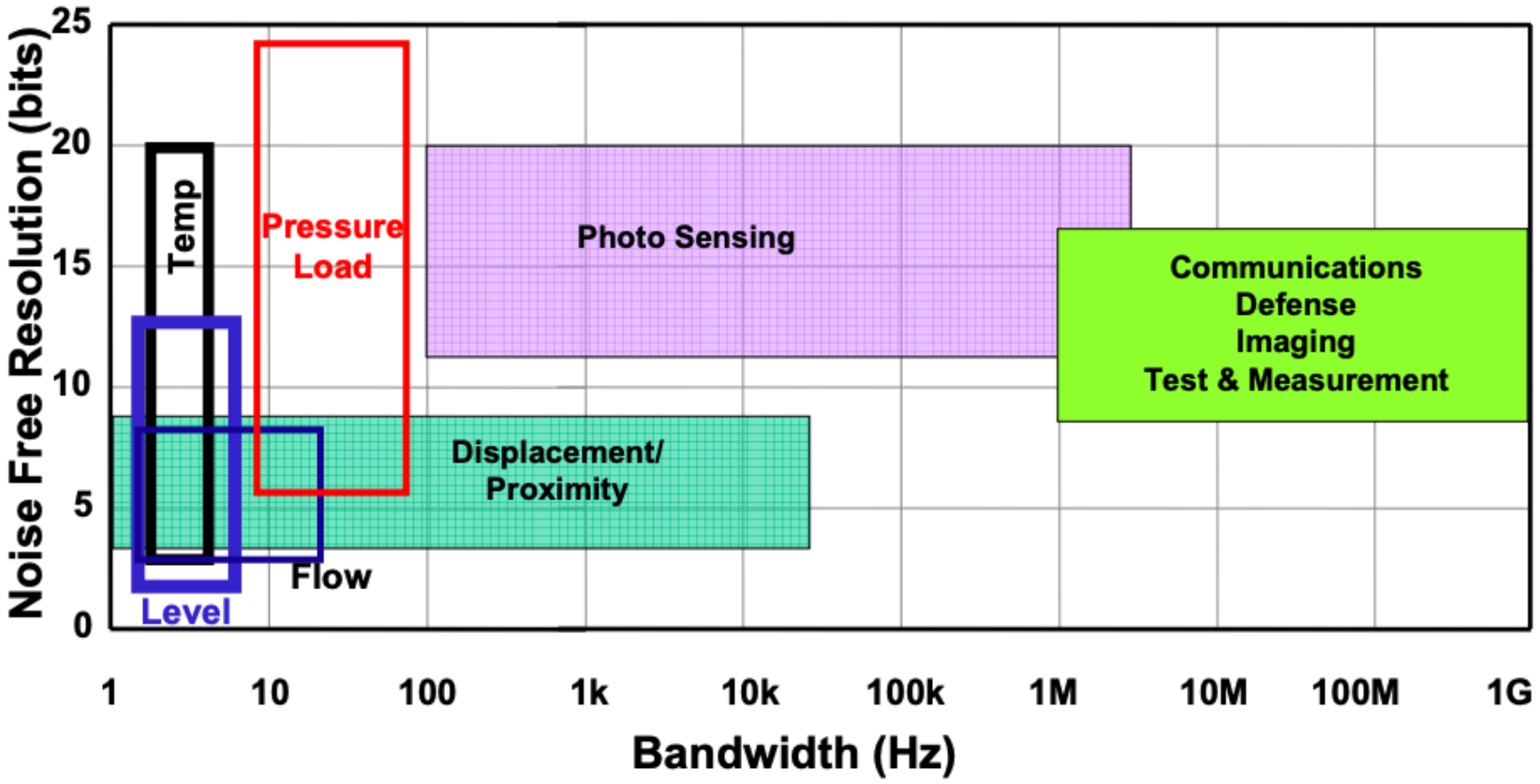
- **Definition**: The sampling rate (or sampling frequency) is the number of times per second that the ADC samples the analog signal. It's typically measured in Hertz (Hz).
- **Impact**: A higher sampling rate means that the ADC takes more samples per second, which generally results in a more accurate signal.

Resolution

- **Definition:** Resolution refers to the number of bits used to represent each sampled value. It's usually expressed in bits (e.g., 8-bit, 12-bit, 16-bit).
- **Impact:** Higher resolution allows the ADC to represent the analog signal with finer granularity.

For example, a 12-bit ADC can distinguish 2^12 (4096) different levels, whereas an 8-bit ADC can only distinguish 2^8 (256) levels. Higher resolution provides more precise digital representations of the analog input but does not directly affect the temporal accuracy of the sampling.

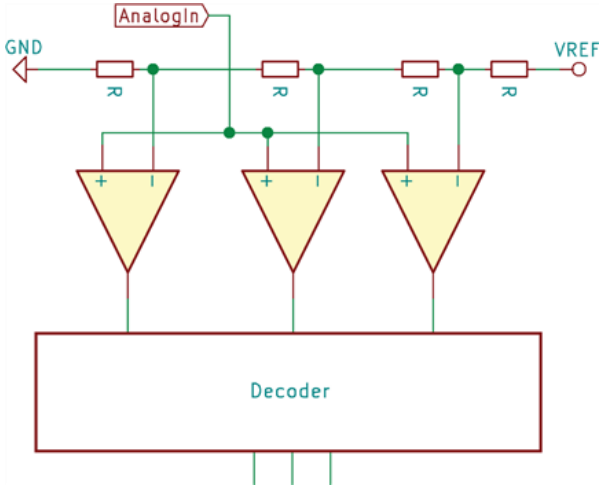
Resolution vs Sampling Time



Types of ADCs

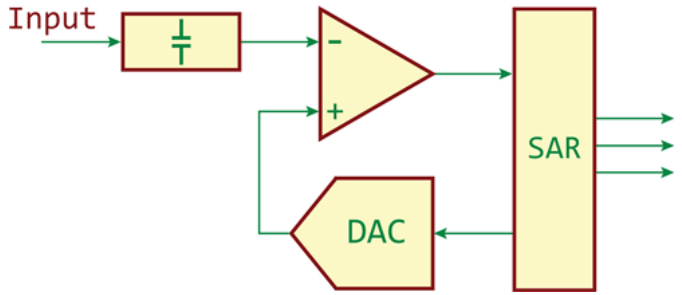
1) Flash ADC

- **Operation**: Uses a parallel array of comparators, each comparing the input signal to a unique reference voltage. All comparisons are done simultaneously, and the outputs are combined to form a digital code.
- **Advantages**: Extremely fast conversion time since it performs the conversion in a single step; ideal for high-speed applications.
- **Typical Uses**: Digital oscilloscopes, high-speed data acquisition, radar systems.



2) Successive Approximation Register (SAR) ADC

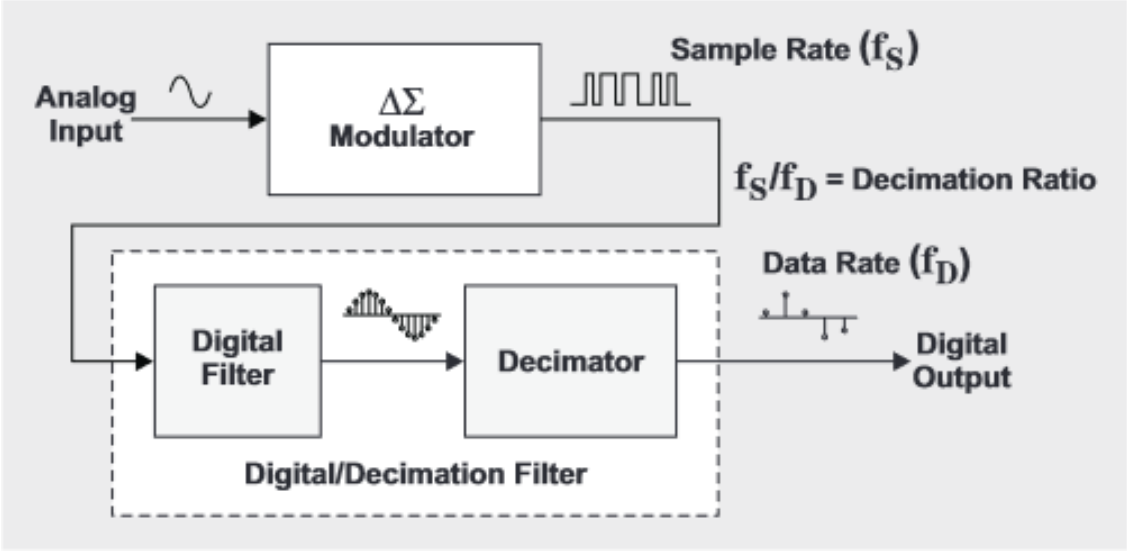
- **Operation**: Uses a binary search algorithm to convert the input voltage into a digital value. The ADC compares the input voltage to a reference voltage in a series of steps, narrowing down the range to determine the closest digital value.
- **Advantages**: Balances speed and resolution; relatively simple and cost-effective.
- **Typical Uses**: General-purpose applications, industrial measurement, data acquisition.



Types of ADCs

Delta-Sigma (ΔΣ) ADC

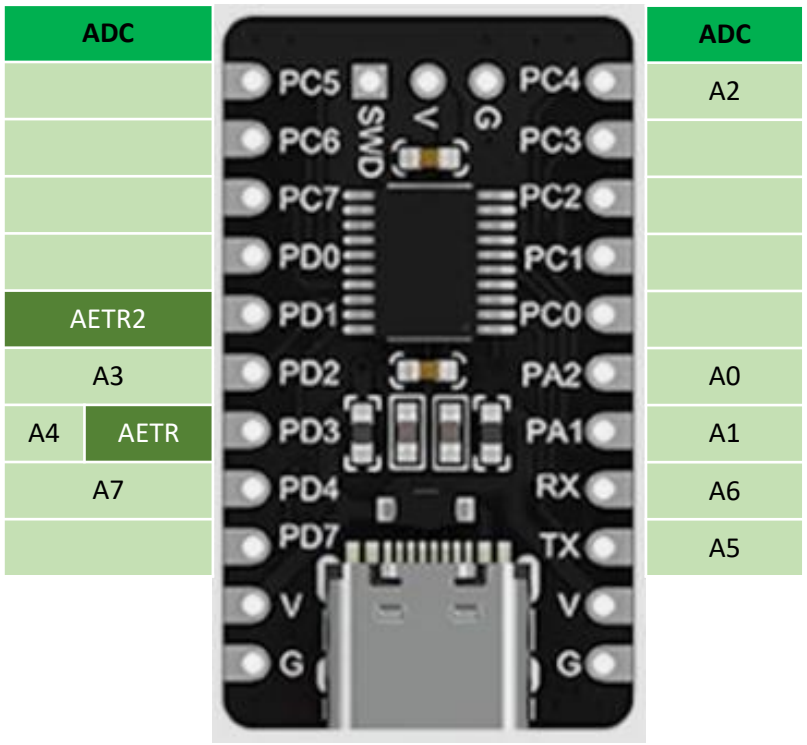
- **Operation**: Oversamples the input signal at a much higher rate than the Nyquist frequency, then uses digital filtering and decimation to achieve high resolution. This process modulates the input signal into a higher frequency and filters it to produce a high-resolution output.
- **Advantages**: High resolution and accuracy; excellent noise performance; good for low-frequency signals.
- **Typical Uses**: Audio processing, precision measurement applications, low-frequency signal acquisition.

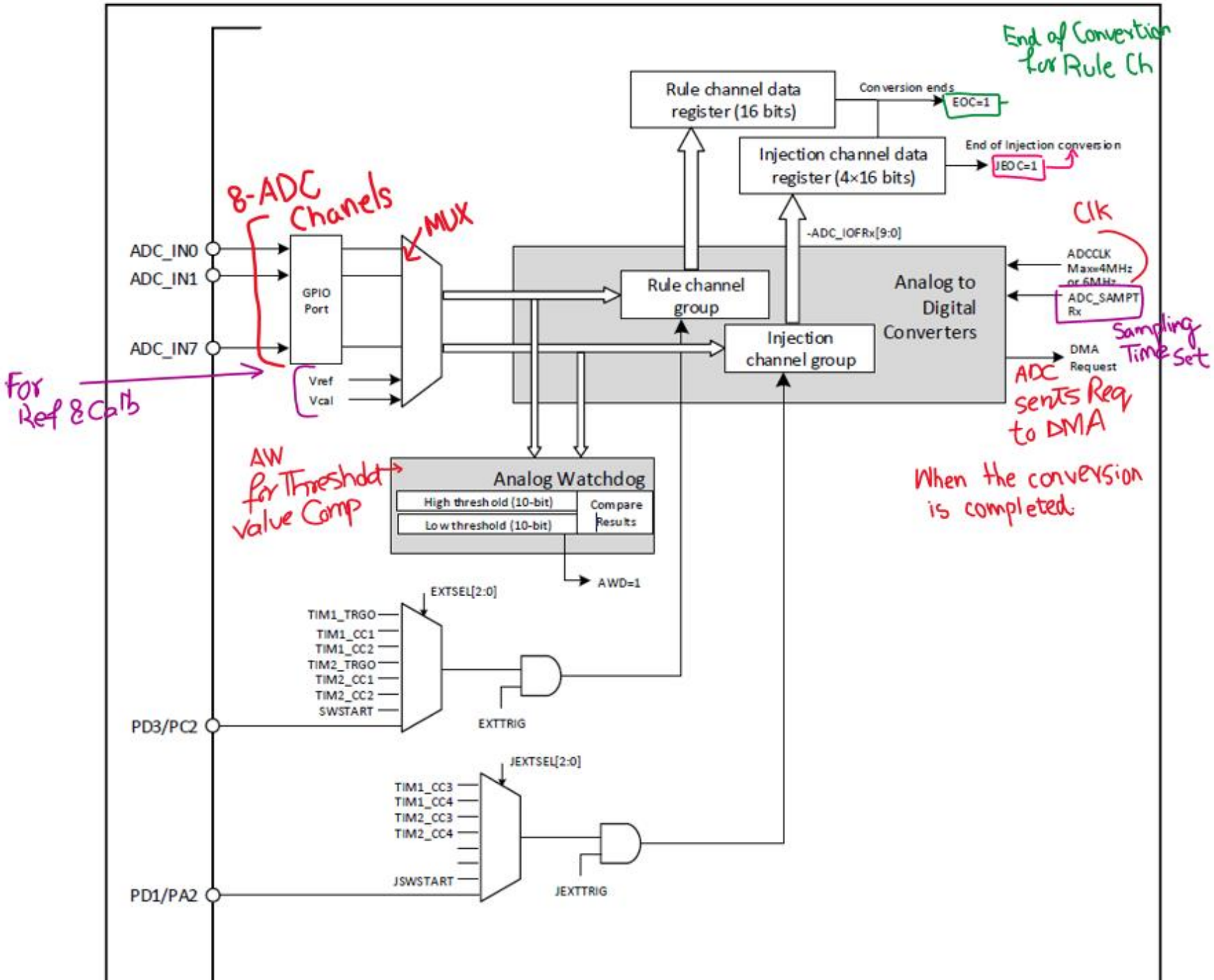


Main Features of ADC in CH32V003

- 10-bit resolution
- Supports 8 external channels and 2 internal signal sources for sampling
- Multiple sampling conversion methods for multiple channels:
  - Single Mode
  - Continuous Mod
  - Scan Mode
  - Trigger Mode
  - Intermittent Mode
- Data alignment modes: left-aligned, right-aligned
- Sampling time can be programmed separately by channel
- Both rule conversion and injection conversion support external triggering
- Analog watchdog to monitor channel voltage, self-calibration function
- ADC channel input range: 0 ≤ VIN ≤ VDDA
- Trigger delay

Default Pin Mapping of ADC




End of Convertion for Rule Ch
Rule channel data register (16 bits)
Conversion ends
EOC=1
Injection channel data register (4×16 bits)
End of Injection conversion
JEOC=1
8-ADC Chanels
MUX
Clk
ADC_IN0
ADC_IN1
ADC_IN7
GPIO Port
-ADC_IOFRx[9:0]
Rule channel group
Injection channel group
Analog to Digital Converters
ADCCLK Max=4MHz or 6MHz
ADC_SAMPT Rx
Sampling Time Set
DMA Request
ADC sents Req to DMA
For Ref & Calb
Vref
Vcal
AW for Threshdt Value Comp
Analog Watchdog
High threshold (10-bit)
Low threshold (10-bit)
Compare Results
AWD=1
When the conversion is completed
EXTSEL[2:0]
TIM1_TRGO
TIM1_CC1
TIM1_CC2
TIM2_TRGO
TIM2_CC1
TIM2_CC2
SWSTART
EXTTRIG
PD3/PC2
JEXTSEL[2:0]
TIM1_CC3
TIM1_CC4
TIM2_CC3
TIM2_CC4
JSWSTART
JEXTTRIG
PD1/PA2

ADC Configuration Flow

1. Module Power up:
2. Sampling Clock
3. Data alignment
4. Channel configuration
5. Calibration
6. Programmable sampling time
7. Read ADC
   - Regular Group Conversion
   - Injected Group Conversion